Controlling Applets' Behavior in a Browser

Vesna Hassler, Oliver Then
Information Systems Institute
Technical University of Vienna
Argentinierstrasse 8/3rd floor, 1040 Vienna, Austria
hassler@infosys.tuwien.ac.at

Abstract

In this paper we discuss methods of protecting Java-enabled Web browsers against malicious applets. Malicious applets involve denial of service, invasion of privacy and annoyance. Since system modification by applets is generally impossible because of the Java security concept, denial of service is of major concern. Invasion of privacy may be caused by applets staying resident in the browser and collecting information about a user. Annoyance may, for example, be caused by advertisment applets that constantly appear on a Web site frequently visited by the user. A general solution to confront such attacks is to have some mechanism within the browser to monitor applets' activities. This mechanism should enable manual or automatic stopping of malicious applets. To illustrate it we present a special applet, called AppletGuard, that allows the user to observe and control the applets in the browser and, based on an applet's properties, stop or suspend the applet, or just warn the user that something dangerous might be going on.

1. Introduction

In this paper we propose a method of protecting Java-enabled Web browsers against malicious applets. Malicious applets involve denial of service, invasion of privacy and annoyance [1]. A general solution to confront such attacks is to have some mechanism within the browser to monitor applets' activities. This mechanism should enable manual or automatic stopping of malicious applets, such as applets that start many threads or use more memory or other resources than allowed. At the time of this writing, we have not seen such a mechanism in any browser implementation yet, which effectively means that all Java-enabled browsers we know are prone to such kind of attacks.

Although the Java security model [3] has significantly improved so that system modification by applets is generally impossible, there are some potential attacks that Java doesn't currently attempt to prevent, such as the abuse of some not necessarily sensitive resources (e.g. CPU cycles, memory and windows) [7]. In other words, there are no resource quotas for untrusted classes. Recovering from denial of service attacks is in most cases not very difficult, e.g. you may have to reboot your machine. However, if the machine has to be available for some critical tasks, denial of service could be very serious. An applet can also perform a degradation of service attack, which means that it significantly reduces the performance of the browser without stopping it [11].

The same type of mechanism that would enable us to control denial of service applets would also give us the possibility to monitor many different kinds of applets' properties, such as the applet name or the time interval within which an applet is active in the browser. For example, some users might like to disable advertisement applets on a Web page they frequently download (annoyance). Or, measuring the time can help detect subtle attacks such as using a machine for an exhaustive encryption key search. The same mechanism could help detect invasion of privacy caused by applets staying resident in the browser and collecting information about the user.

To illustrate this protection mechanism we present a special applet, called AppletGuard, that allows the user to observe and control the applets in the browser. Based on an applet's properties, the browser can stop or suspend the applet, or just warn the user that something dangerous might be going on. The AppletGuard protects against denial of service attacks caused by applets that use excessive resources, or against applets with specific properties. The actual implementation uses some security

bugs in the older versions of JDK (Java Development Kit [2]). However, our approach does not depend on JDK security holes, but is rather generic. In the last section we discuss the possibilities to implement the described mechanisms based on the customizable parts of the Java security, namely Class Loader and Security Manager.

2. Browser protection mechanisms

A set of protection mechanisms as part of the browser, with special privileges to access all threads running in the browser, would enable the user to

- monitor all applets' activities in the browser,
- manually or automatically stop applets based on specific properties, and
- stop all applets s/he does not want to run longer after leaving their Web page.

In the examples given later we measure the number of threads and stack frames used by an applet as well as check the names of applets, but it could equally be possible to use the mechanism to controls applets based on other properties, such as

- the time an applet is active in the browser, and/or automatically stop all applets hanging around for more than a predifined time interval, or
- the number of windows started by an applet, or
- the applet's memory consumption, or
- the applet's CPU time, or
- the applet's source (i.e. URL, see also [8]), or
- the applet's digital signature (i.e. certificate).

If based on any of the properties an applet appears "suspicious" to the browser, it should be possible to stop or suspend selected threads or the whole applet. In case of suspension, the browser should ask the user if s/he wishes to continue execution.

In the sequel we discuss the possibilities of implementing these mechanisms. Our discussion is based on the customizable parts of the Java security model, namely Class Loader and Security Manager.

2.1. Applet Class Loader

The `ClassLoader` class in the `java.lang`, package is an abstract class that provides the programming interface and partial implementation for all Java class loaders. There are usually several class loaders in operation; the applet class loader installs each applet in a separate name space (applets from the same page and source may share a class loader). Java's security rules prohibit applets from creating class loaders.

The applet class loader could be modified to prevent from loading all applets whose class name contains a predefined substring or comes from a predefined sources. The substrings to watch for would be specified by the user in the user's preferences, or, for example, by klicking on the applet's area in the browser and selecting "Always prevent from loading" from a pop-up menu. A similar approach is described in [8] where the authors actually replaced the applet class loader in the browser class library by their version.

2.2. Security Manager

The `SecurityManager` class in the `java.lang`, package is an abstract class that provides the programming interface and partial implementation for all Java security managers. It is meant to be defined by the browser through subclassing to perform run time checks on dangerous methods. In an application the security manager can be set only once. The security manager can determine which classes are involved in the current request and which class loader is responsible for that class. If the responsible class loader is the applet class loader, it knows that the request comes from the untrusted code.

If the browser wishes to control, for instance, the number of threads that can be created by an applet, the security manager's method `checkAccess(ThreadGroup)` which is invoked when a new child thread or child thread group is created, should check how many threads have already been started by the applet before allowing it to start a new one. The number of threads might be defined by the user's preferences, or by the browser's default values. The browser might also suspend the applet and ask the user whether to permit it to start a new thread. However, the problem with this method is that the security manager doesn't know whether a thread is being created or an entire thread group is being destroyed [6].

Another example is to limit the memory used by an applet. This could be achieved by counting the stack frames used by an applet or by a thread group. It could be partly implemented in such a way that the security manager's method `checkAccess(ThreadGroup)` computes the number of stack frames used by a thread group or by an applet and, if the number were higher than allowed (by either the user preferences or the browser default values), it would not permit starting a new thread. However, that could not prevent an existing thread from using more memory (stack frames) than allowed since the

security manager does not have all the information it needs to do resource tracking (i.e. it is stateless).

To prevent such attacks, some monitoring mechanism should periodically check the number of threads started by an applet, and the threads' status. Such mechanism cannot be implemented by using the security manager approach only. According to the JavaSoft's security architects, it would be a very difficult task to provide a generic way of resource tracking without "slowing everyting down" [9]. Similarly, a monitoring mechanism would be necessary to periodically check for open threads belonging to presumably no longer active applets.

3. AppletGuard

In this section we illustrate some of the browser protection mechanisms. For this purpose we present a special applet, called AppletGuard, that allows the user to observe and control the applets in the browser and, based on an applet's properties, stop or suspend the applet, or just warn the user that something dangerous might be going on. The AppletGuard protects against denial of service attacks caused by applets that use excessive resources, or against applets with specific properties. The source code is available from [12].

The idea to write the AppletGuard is based on AssassinApplet from [1]. The actual implementation uses some security bugs in the older versions of JDK (Java Development Kit [2]). The bugs allow each applet to access, and even stop, the threads not belonging to its group. It can be efficiently misused with the older versions of the Netscape browser (2.x and 3.x), but we use it to protect the browser against misbehaving applets. Netscape 4.x does not allow an applet to access the thread groups other than its own. For the current status of the JDK security see [3].

3.1. Java threads

Java-enabled browsers can start multiple threads of execution (multithreading) which also means that several applets can run in parallel within a browser. An applet has at least one thread (the applet itself), but it can also start new threads to accomplish various tasks. The idea our AppletGuard is based on is to install a thread that monitors and controls all other threads in the browser.

Each Java thread in the browser is a member of a thread group [5]. Some threads are automatically produced by the browser when it is started, and some threads are user or applet threads. Most browsers allocate a thread group for each applet [4]. If a thread is started by an applet it is automatically assigned to the applet thread group called applet-CLASSNAME.class (Figure 2). In other words, all threads started by the applet will belong to this group. Each thread group has a parent group, which yields a tree-like structure. An example thread hierarchy in the browser is shown in Figure 1.

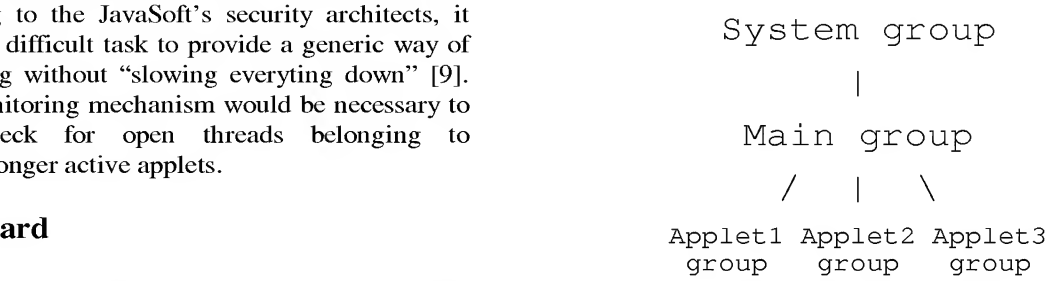

Figure 1: An example thread hierarchy

In Sun Microsystems' security policy for JDK 1.0.*, and consequently in all browsers that are based on it (like Netscape 2.*), there is a thread access bug: each applet can access all threads from other thread groups, i.e. not only its own threads. This bug can be misused to write a malicious applet that can monitor and even stop all the threads in the browser (i.e. all System group threads) [1], as well as count the stack frames (memory) used up by a thread.

In Netscape 3.*-JDK 1.1.* the major thread bug was removed, but we discovered that there is still a possibility to gain control over all applet threads in the browser. To achieve this it is necessary to overwrite any applet function that has a thread in the Main thread group (e.g. `paint()`). With this bug it is possible to monitor and stop other applets' threads, but we could not find out how to count their stack frames.

3.2. AppletGuard's functionality

In our experimental implementation it is very important the AppletGuard to be the first applet loaded after starting the browser. In this way it cannot happen that the browser or even the machine freezes due to a denial of service attack already taking place. The AppletGuard's basic functionality can be described as follows:

1. Spawn a monitoring thread.
2. Go one level up in the thread hierarchy and save this thread group's reference.
3. From this thread group downwards, scan all threads.
4. For each scanned thread, display its status and check if it exhibits malicious behavior (i.e. starts to many

threads or uses up to many stack frames) or has specific properties.

5. Stop misbehaving threads.

The steps 2-5 can be repeated for all thread hierarchy levels. With this approach, most denial of service attacks coming from other applets can be timely discovered and stopped. Specifically, it is possible to

- *prevent an applet from starting too many threads*: the status of all threads is displayed in the AppletGuard window so that they can be stopped manually or automatically
- *prevent an applet from using to much memory*: the number of stack frames used by an applet is also shown in the AppletGuard window, so that based on this information the malicious applet can be stopped
- *prevent an applet to stay resident in the browser after the user has left the applet's Web page*: the status of all applets' threads is shown in AppletGuard window, so it can be easily noticed if there are some threads hanging around belonging to the supposedly stopped applets
- *stop all applets containing a predefined substring in the name*: in this way it is possible to disable e.g. advertisement applets that are a nuisance to the user

If the user prefers to stop selected applets or threads *manually*, only a warning is shown in the status display each time an applet's properties match the properties selected to be watched for (e.g. number of threads or name). If the user wants the applets to be stopped *automatically*, a status message informing him/her about an action is shown in the status display.

3.3. AppletGuard's options

In Figure 2 the main window of the AppletGuard is shown. We loaded the AppletGuard immediately after starting the browser (it can be e.g. on the home page). Figure 2 shows the intial status. We can see that the AppletGuard starts two threads, Thread-1 and AppletGuard2, both belonging to the AppletGuard2 applet.

If we push the Options button in Figure 2, the options window (Figure 3) pops up. We can select manual or automatic killing (i.e. stopping) of applets. If we select automatic killing, all applets starting more than 7 threads and using more than 7 stack frames will be stopped automatically. If we select manual killing, we can delete selected applets and threads in the main window by pushing the Stop button.

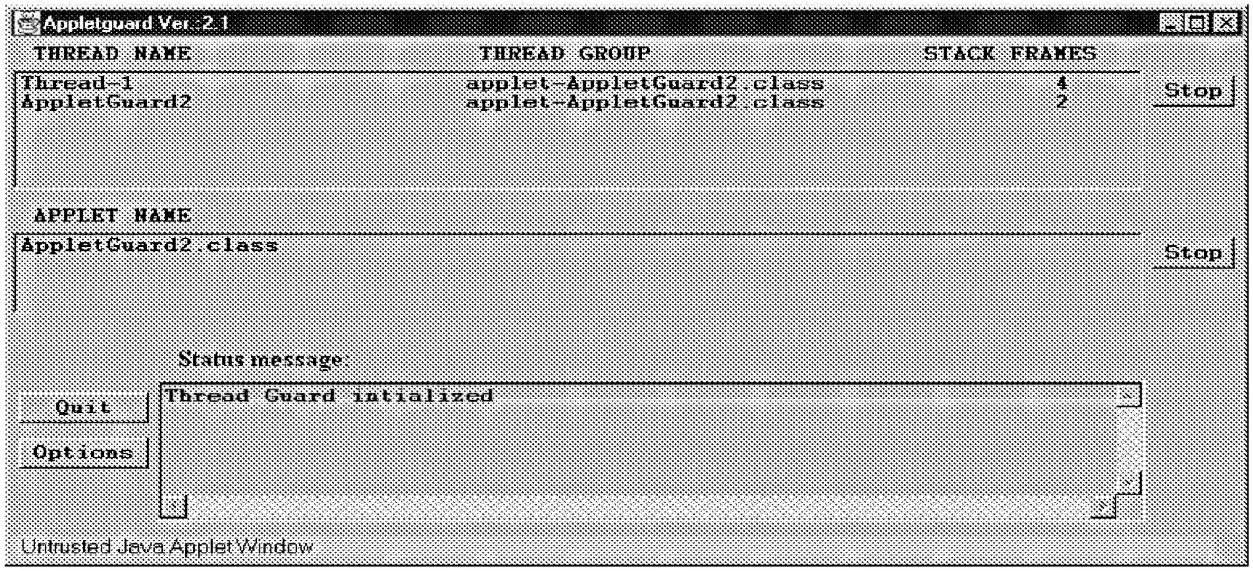

Figure 2 : AppletGuard main window

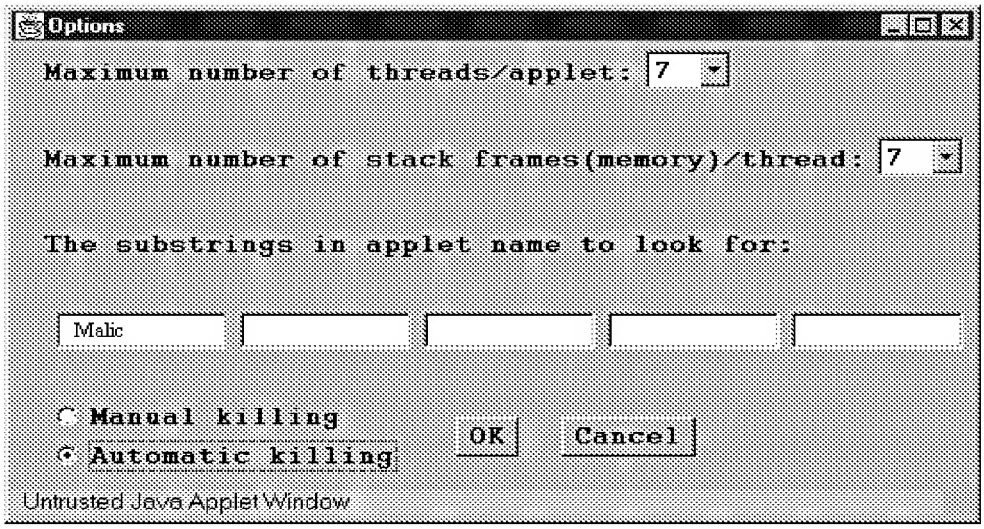

Figure 3: AppletGuard options window

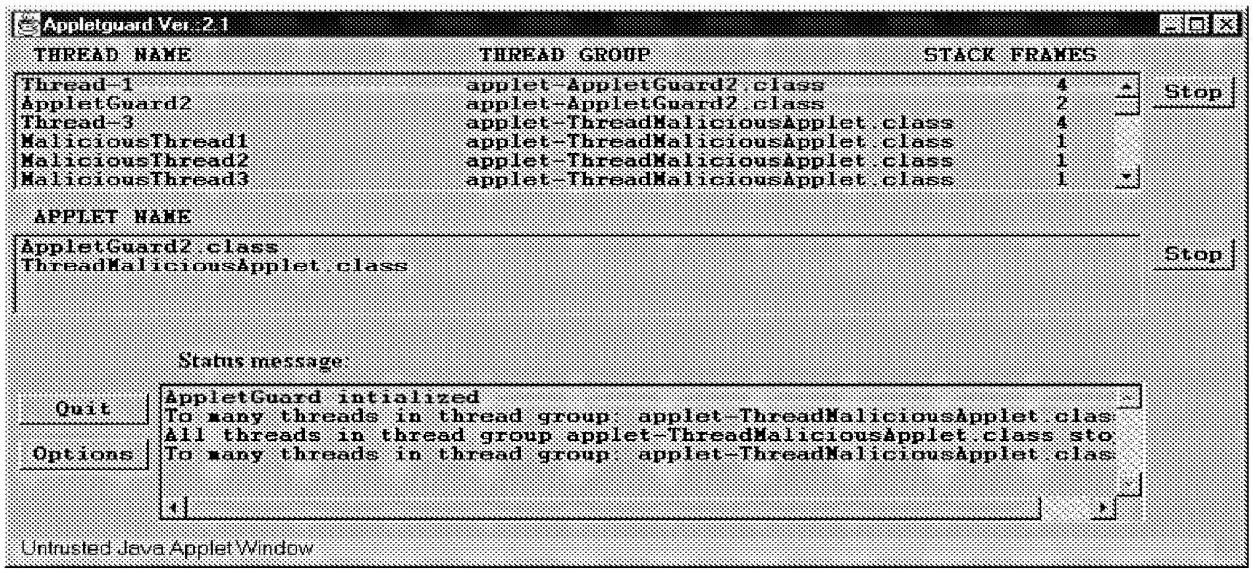

Figure 4: AppletGuard in action: Applet with too many threads is stopped

3.4. Example 1

In Figure 4 we can see how the AppletGuard works when the browser is attacked. We select the option for automatic killing in the options window (Figure 3) and load the ThreadMaliciousApplet that tries to start 20 idle threads. Since we don't allow more than 7 threads to be started by any applet, ThreadMaliciousApplet is automatically killed with the status message "*All threads in thread group applet - ThreadMaliciousApplet.class stopped*" (second status message in Figure 4).

3.5. Example 2

In the second example we select the option for manual deleting and load the same misbehaving applet again. The AppletGuard displays it in the topmost status frame: Thread-3 and many MaliciousThreads (Figure 4). Since this attack is not so serious, i.e. it does not freeze the browser, we are able to select (using the mouse) the applet *ThreadMaliciousApplet.class* in the middle status frame and stop it by pushing the Stop button. However, some more serious attacks could disable any kind of manual intervention, so it is recommendable to enable automatic killing.

3.6. AppletGuard should be a part of the browser

Unfortunately, this implementation of the AppletGuard cannot be used for serious protection. Suppose a malicious applet knows about the existence of the AppletGuard. The only thing the malicious applet has to do in order to disable the AppletGuard is to first stop all applets running in the browser using the same technique we are using to protect the browser. The newest version of the Netscape browser (4.x) does not allow any applet to access other groups' threads anyway.

4. Summary and conclusions

In this paper we discuss some mechanisms to protect a Java-enabled browser against malicious applets, such as the applets causing denial of service attacks, or applets that are annoying the user. The mechanisms are based on a method for monitoring and controlling Java applets and their threads running in the browser. Such mechanisms should be part of each Java-enabled browser, but we have not seen any implementation so far. To illustrate our ideas, we wrote a special applet called AppletGuard which enables the user to monitor and, if necessary, stop selected threads and applets. We tested the AppletGuard with older versions of the Netscape browser (2.x and 3.x). It does not offer perfect protection since the AppletGuard has only the privileges of a normal applet. We also discussed the possibilities to implement the mechanism based on the Java class loader and Java security manager, and realized that it would be difficult since they do not offer a straightforward way for resource tracking. We hope to see similar mechanisms as a part of Java-enabled browsers in the near future.

We started to implement the AppletGuard in September 1997. At the time of finalizing the camera-ready version of the paper (September 1998) we found out that the HotJava browser (version 1.1.4) from Sun Microsystems implements a mechanism to monitor and selectively kill applets. However, there is no possibility to configure the browser to watch for applets having specific properties, as with the mechanism described in the paper. Additionally, we unfortunately discovered a security flaw in the browser: with our AppletGuard we could kill all threads from the HotJava thread group, including the threads implementing the thread monitoring mechanism (ThreadCountApplet and ThreadListApplet). They couldn't even be loaded again, although it was generally impossible for the AppletGuard to access other applets' threads. We then tried to kill the AppletGuard from the HotJava browser. However, although the HotJava thread monitor showed that the AppletGuard had been killed, it wasn't quite true, since we then killed the HotJava monitoring threads by the same allegedly "dead" instance of AppletGuard.

References

[1] McGraw, G., E.W. Felten, *Java Security. Hostile Applets, Holes, and Antidotes*, John Wiley & Sons, Inc., 1997

[2] Java Development Kit, Sun Microsystems, http://www.javasoft.com/products/jdk/1.1/index.html

[3] Java Security, Javasoft, http://www.javasoft.com/security/

[4] The Java Tutorial, http://www.infosys.tuwien.ac.at/NEW/Services/Docu/Java/Tutorial/

[5] Oaks S., H. Wong, *Java Threads*, O'Reilly, 1997

[6] Glenn Vanderburg et al., *Maximum Java 1.1*, Sams.net Publishing, 1997

[7] Michael Morrison et al., *Unleashed Java 1.1*, Sams.net Publishing, 1997

[8] Dirk Balfanz, Ed Felten, "A Java Filter", Technical Report 567-97, Department of Computer Science, Princeton University, October 1997

[9] Java Security Archive, http://java.sun.com/security/hypermail/java-security-archive/index.html

[10] Vesna Hassler, Oliver Then, "Controlling applets' behavior in a browser", Technical Report TUV-1841-98-03, Distributed Systems Group, Technical University of Vienna, 1998

[11] Drew Dean, Edward W. Felten, Dan S. Wallach, and Dirk Balfanz, "Java Security: Web Browers and Beyond", *Internet Beseiged: Countering Cyberspace Scofflaws*, Dorothy E. Denning and Peter J. Denning, eds. ACM Press (New York, New York), October 1997

[12] Oliver Then, " Monitoring applet protecting against denial of service attacks", Informatikpraktikum II, Distributed Systems Group, Technical University of Vienna, February 1998, http://www.infosys.tuwien.ac.at/Courses/Results/Praktika/AppletGuard/